Construction of Short Block Length Irregular Low-Density Parity-Check Codes

Aditya Ramamoorthy and Richard Wesel
Department of Electrical Engineering, UCLA
Los Angeles, CA 90095-1594
{aditya r,wesel}@ee.ucla.edu

***Abstract*— We present a construction algorithm for short block length irregular low-density parity-check (LDPC) codes. Based on a novel interpretation of stopping sets in terms of the parity-check matrix, we present an approximate trellis-based search algorithm that detects many stopping sets. Growing the parity check matrix by a combination of random generation and the trellis-based search, we obtain codes that possess error floors orders of magnitude below randomly constructed codes and significantly better than other comparable constructions.**

I. Introduction

Low-Density Parity-Check (LDPC) codes have been the subject of intense research lately because of their linear decoding complexity and capacity-achieving performance. Irregular LDPC codes have emerged as strong competitors to turbo-codes. Constructions of LDPC codes are usually random in nature. A few algebraic constructions are known but these are typically for regular codes that lack the capacity-achieving ability of irregular codes.

Using concentration theorems proved in [1], one can show that in the limit of infinite block length, the performance of the code will tend to cluster around the mean value. The analysis proceeds by showing that for large block length the local neighborhood of a node is tree-like and hence the belief propagation is exact. However for short block lengths, the local neighborhood of the nodes is inevitably non-tree-like and thus there is considerable variation among codes from a given degree-distribution ensemble.

Since cycles in the Tanner graph cause the sub-optimality of belief propagation, one approach [2],[3] has been girth-conditioning, where one tries to remove as many short cycles as possible. However such conditioning is not easy to perform for high-degree variable nodes, in fact the highest degree considered in [2] was 3.

Di et. al [4] introduced the concept of "stopping sets", which cause the iterative decoder to fail when operating over the BEC. Tian et. al [5] [6] introduced a metric called ACE (approximate cycle extrinsic message degree (EMD)) and designed codes that reduced small stopping sets. They found that such codes also had a good error floor performance over the AWGN channel, beating randomly generated codes by several orders of magnitude.

This work was supported by Texas Instruments and the state of California through UC MICRO grant 03-093 and NSF grant CCR-0209110.

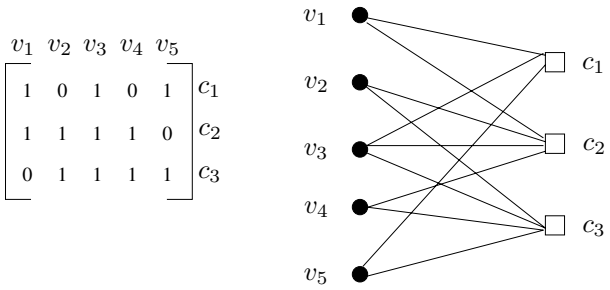

Fig. 1. Tanner graph and Parity Matrix of a (5,2) code

Under belief propagation decoding, the low-degree variable nodes take longer to converge and have a higher probability of being in error. Thus, both of the above techniques concentrate on conditioning the low degree nodes. This work presents a greedy code construction technique that employs code conditioning on "all the variable nodes" and statistically avoids stopping sets.

Section II provides a brief overview of stopping sets and introduces an equivalent definition of stopping sets in terms of the parity check matrix. Section III outlines the new code construction algorithm and also explains how it can be used along with the method of [5] to obtain codes with even better error floors. It also explains why it is hard (at least for a greedy construction) to avoid all stopping sets of a given size (we are unaware of an algorithm that accomplishes this for an arbitrary degree distribution and block length). Section IV presents a discussion of the results and explores the possibilities for future work.

II. The Code Construction Technique

Fig. 1 shows the Tanner graph and the parity matrix of an example (5,2) code. The equivalence between the parity check matrix (H) and the Tanner graph of a code is well known. Thus, we shall use the terms "variable node" and "column" interchangeably.

Definition 1: [4] ***Stopping Set - Graph Perspective***
A set S of variable nodes is said to form a stopping set if all its neighboring check nodes are connected to S at least twice. ■

For example, the variable nodes v_1, v_2 and v_3 form a stopping set in Fig 1. Another way of defining a stopping set in terms of

the parity check matrix, which is more convenient and intuitive for the presentation of our algorithm, is given below.

Definition 2: ***Stopping Set - Parity Check Matrix Perspective***

First we define a function that will be used throughout the paper.

$$f_{sc}(\alpha) = \sum_i I(\alpha[i] = 1) \quad (1)$$

Where α is a column vector and $I(x)$ is the indicator function. Thus f_{sc} counts the number of 1's in a column vector, *sc* in the subscript denotes "singly-connected".

Consider a subset S of the columns of the parity check matrix of size $m \times n$. Let $\Delta = \sum_{i \in S} v_i$, where the sum is over the real field (not over GF(2)). The set S forms a stopping set if $f_{sc}(\Delta) = 0$. ■

For example in Fig. 1 the set of variable nodes v_1, v_2 and v_3 is such that $v_1 + v_2 + v_3 = [2 \;\; 3 \;\; 2]^T$ which does not have a 1 in any component. So, it forms a stopping set.

Observe that all codewords are also stopping sets and the size of the minimum stopping set in a code is a lower bound on the minimum distance of the code. It is well known that the stopping set spectrum of a code completely determines it's performance over the BEC [4]. While stopping sets do not behave in exactly the same manner over the AWGN channel as over the BEC, variable nodes with poor reliabilities can be considered similar to erasures and thus removing small stopping sets should improve the error floor performance of the code. This provides the main motivation for this work. This perspective was validated in [5].

III. The Code Construction Algorithm

Let the parity check matrix of a code of rate $R = \frac{k}{n}$ be denoted by H of size $m \times n$ where $m = n - k$. Our objective is two-fold. The code rate needs to be R, requiring H to be full-rank and the number of small stopping sets needs to be low. Let $H = [H_m | H_k]$, where H_m is of size $m \times m$ and H_k is of size $m \times k$. The algorithm ensures that H_m is a full rank matrix. Since low degree nodes are more likely to form small stopping sets the algorithm proceeds by generating columns in increasing order of degree. The full description of the algorithm is given in Fig. 2. The matrix is full-rank since we continue to generate new vectors at random until we find a full basis that also passes the *Stopping-Set-Check*. The *Stopping-Set-Check(Type,v,H,d)* function has four different inputs.

1) *Type = ('E')xhaustive or ('A')pproximate* is an input that decides whether the algorithm is Exhaustive or Approximate in nature. (we explain in more detail below)
2) v is the new variable node
3) H is the "current" parity check matrix
4) d is the depth parameter (again, more details follow).

A. Stopping-Set-Check - (E)xhaustive Mode

Suppose that *Stopping-Set-Check* is used in the (E)xhaustive mode and the depth parameter is set to be d_1 (say). The algorithm is described by,

```
for (i = 0; i < n; i++)
begin
Step 1:
   Generate v_i at random according to deg(v_i);
   if i < m (i.e., v_i is a parity bit)
     if v_i ∈ SPAN(H_m)
      goto Step 1;
     else
       if Stopping-Set-Check('A',v_i,H,d)
       Add v_i to H_m (i.e., passed both tests)
     else
       goto Step 1
   else
     if Stopping-Set-Check('A',v_i,H,d)
       Add v_i to H_k
     else
       goto Step 1
   end
end
```

Fig. 2. **Code Generation Algorithm**

Stopping-Set-Check('E',v,H,d)

$$= 1 \quad \text{If } [H|v] \text{ is free of stopping sets of size} \leq d_1 \quad (2)$$
$$= 0 \quad \text{otherwise}$$

$[H|v]$ denotes the new Tanner graph formed by adding v to H. The algorithm outputs a Tanner graph that is free of all stopping sets of size $\leq d_1$ on termination. This is because on the completion of $(j-1)^{th}$ stage of the algorithm, the set $v_0, v_1, ..., v_{j-1}$ is free of all stopping sets of size $\leq d_1$. Therefore, any stopping set that is created at stage j, will involve the new node v_j. The nature of the search is exhaustive, thus v_j will be kept only if it passes the test. The conclusion follows inductively.

Unfortunately we are unaware of any algorithm that can perform the above task without resorting to a brute force check of the $\sum_{i=1}^{d_1-1} \binom{j-1}{i}$ possible combinations of variable nodes that v_j can form a stopping set with. Once j is fairly large, the large complexity quickly renders the code generation process intractable. Note that we are not claiming that such an algorithm does not exist. However, we were unable to find one.

B. Stopping-Set-Check - (A)pproximate Mode

Since it is difficult to eliminate all stopping sets of a certain size, we resort to good heuristics that eliminate as many stopping sets as possible. A heuristic that we tested and found to perform well in practice is presented here.

As in [5], our technique is basically a Viterbi-like search algorithm for each new variable node that is added to H. The search extends for up to a specified number of trellis stages, which depends on the degree of the new node. The trellis stages are numbered 0 - $(d-1)$. Even/Odd stages correspond to variable/check nodes respectively. A trellis branch corresponds to a connection between nodes in the Tanner graph.

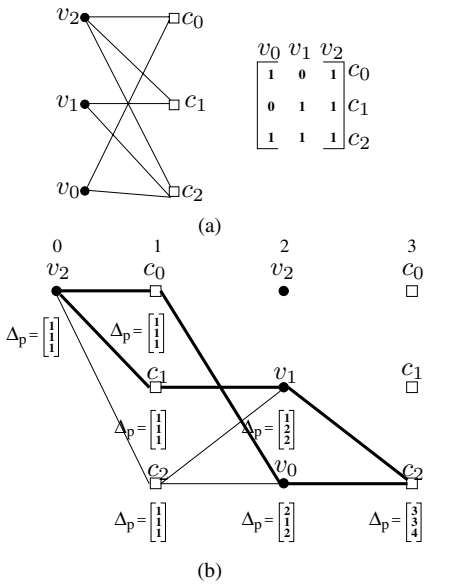

Fig. 3. (a) Example where v_2 is being tested (b) Flow of the algorithm on the trellis (bold lines represent survivor paths)

The algorithm is demonstrated in Fig. 3(b) for the example shown in Fig. 3(a). A new node (e.g. v_2 in Fig. 3) is generated at random according to it's degree and in accordance with the overall degree distribution and a trellis-based search is performed to detect the stopping sets in which it participates. Paths/Cycles with repeated nodes are disallowed. Let the variable node for which we are performing the search be denoted v_{root}, the number of trellis stages $d_1(v_{root})$ and the current node be denoted v_t or c_t (depending on whether it is a variable node or a check node). We need the following definitions,

- Δ_p, the path metric, is an m-dimensional vector containing the sum of the variable nodes (columns) that participate in the path p.
- β_p is the number of singly-connected check nodes in Δ_p. i.e. $\beta_p = f_{sc}(\Delta_p)$.
- β_c is the current minimum of the number of singly-connected nodes in any cycle containing v_{root} found so far. It is initialized to ∞.
- γ_p is the minimum-path-metric threshold to be satisfied in the trellis search
- γ_c is the minimum-cycle-metric threshold to be satisfied in the trellis search

1) **Minimum Path Metric** (β_p) - The search algorithm maintains a list of the current paths from the root node to the current trellis stage. The path metric for a given path p is defined to be,

$$\Delta_p = \sum_{i \in V_p} v_i,$$

where V_p is set of variable nodes participating in p

$$\beta_p = f_{sc}(\Delta_p) \tag{3}$$

The minimum of β_p's over all paths i.e. $\min_{\forall p}\beta_p$ is also computed and stored in memory.

2) **Minimum Cycle Metric** (β_c)- At any point in the search, a merge at a particular node in the trellis indicates the existence of a cycle. Suppose that the cycle is composed of two paths p_1 and p_2 that merge at either a check node c_t or a variable node v_t. The number of singly-connected check nodes in the cycle needs to be counted. This update is handled differently depending on whether the merge is at a variable node or a check node.

a) **Check Node - Merge Update**

$$\beta_c = \min(f_{sc}(\Delta_{p_1} + \Delta_{p_2} - v_{root}), \beta_c) \tag{4}$$

The contribution of the root is subtracted so that it is considered only once.

b) **Variable Node - Merge Update**

$$\beta_c = \min(f_{sc}(\Delta_{p_1} + \Delta_{p_2} + v_t - v_{root}), \beta_c) \tag{5}$$

For the variable node update the merge node v_t also needs to be considered since it forms part of the cycle. Again, the contribution of the root is subtracted.

The merge update equation computes the number of singly-connected check nodes in the cycle. As discussed in Definition 2, if this number is greater than or equal to 1, then the cycle is not a stopping set.

If at any trellis stage $\min_{\forall p}\beta_p < \gamma_p$ or $\beta_c < \gamma_c$ a failure is declared and the variable node is regenerated.

C. Explanation of the Approximate Search

To be able to perform a Viterbi-like search, the algorithm needs to decide which particular path to choose as a survivor at a merge. There are two possibilities.

a. **Minimum Cycle/Path Metric Violation**
In this situation, at least one set of variable nodes that violates either the minimum cycle metric (γ_c) or the minimum path metric (γ_p) has been found. So we declare a failure and regenerate the root node. We do not need to make a decision about the survivor path.

b. **Minimum Cycle/Path Metric Pass**
Here a decision is required on which path is more likely to cause a cycle/path metric violation deeper in the trellis, otherwise the total number of paths will grow exponentially with depth.
If the objective is to find all stopping sets below a certain size, it is not proper to choose one path over the other. It can happen that a particular path that has a higher metric at the current stage causes a violation deeper in the trellis whereas the path with lower metric does not. However for the sake of reduced complexity the survivor path is chosen to be the one that has the lower number of singly-connected nodes at the merge node (i.e. the lower metric). This is intuitively the right choice since the path with the lower number of singly-connected nodes is more likely to form a stopping set later on in the search. Ties are broken by choosing the path highest in the lexicographic ordering.

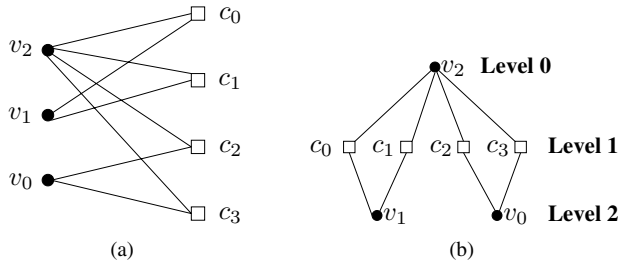

Fig. 4. (a) v_0, v_1 and v_2 form a stopping set, but they are not connected by "one" cycle (b) A tree-based descent considering combinations of v_2 and variable nodes (v_1, v_0) at Level 2 can detect this stopping set.

To see a working example consider Fig. 3 where v_0 and v_1 exist in the graph and a new node v_2 is tested. At the third trellis stage a cycle is formed that violates the minimum cycle metric. The bold lines represent the survivor paths of the Viterbi merge at stage 2. In fact this particular cycle is also a stopping set.

Of course there are other types of stopping sets that cannot be detected by this algorithm since they do not form a single cycle (for an example see Fig.4). One way to get around this problem would be to perform a tree-based descent from the root node and consider different sets of variable nodes at all the even levels and check whether they form a stopping set. However, in the worst case, this would involve considering all possible combinations of variable nodes.

The setting of the thresholds γ_p, γ_c and the number of trellis stages $d(v)$ needs to be discussed. Ideally in the "(E)xhaustive" mode, the setting $(\gamma_p = 1, \gamma_c = 1)$ would suffice to rule out all undesirably small stopping sets. However in the "(A)pproximate" mode, different settings of the thresholds based on the degree of the variable nodes need to be tried. Typically we can use large values of $d(v)$ for low degree variable nodes, since in the initial phase of the code generation process there are only few nodes in the graph and thus finding cycles/paths that violate the cycle/path metric threshold is hard. We mention that this is a weakness of the method that is discussed in more detail in the next subsection. For the medium and high-degree nodes the depth parameter $d(v)$ needs to be reduced since at this stage of the process there are many nodes in the graph and it becomes progressively more difficult to satisfy the path/cycle metric criteria. The γ_p and γ_c parameters are also tied to the degrees of the nodes. γ_p needs to be small for low degree nodes since they anyway have a low number of 1's in their column representation. Medium and high degree nodes can satisfy a higher threshold.

Note that the complexity of running the *Stopping-Set-Check* algorithm is always upper-bounded by $(\max_v \text{degree}(v) - 1) \times (\max_v d(v)) \times (n-1)$, because the maximum degree of a variable node is $\max_v \text{degree}(v)$, the maximum number of trellis stages is $\max_v d(v)$ and the maximum number of variable nodes in the graph is n. However one might need to generate multiple column vectors before one that satisfies the *Stopping-Set-Check* is found. Overall the complexity remains manageable for designing codes of desired block lengths.

D. Improving the Approximate Search

If the "Code Generation Algorithm" uses the *Stopping-Set-Check* as discussed above (in the approximate mode) there are certain problems associated with the low degree nodes. A wrong survivor path decision at a trellis merge (explained in Section III-C(b)), is liable to cause more problems at the low degree nodes rather than high-degree ones since any stopping sets that are left undetected will consist only of low degree nodes. On the other hand, the above algorithm is much more accurate at the medium-to-high degree nodes once the graph has been grown to some extent.

Tian et. al [5] proposed a code construction method where they designed codes that satisfied a metric called the "Approximate Cycle EMD (ACE)".

Definition 3: ***Approximate Cycle EMD (ACE)***
The ACE of a length $2d$ cycle is given by $\sum_i (d_i - 2)$, where d_i is the degree of the i^{th} variable node in the cycle. ■

Their construction guarantees that all cycles in the code of size $\leq 2d_{ACE}$ have an ACE value $\geq \eta$. They found that the codes designed using this criterion had substantially lower error floors than randomly constructed codes. The main drawback of their technique is that it completely ignores high degree nodes. Once the degree of the new variable node $\geq (\eta + 2)$, then it is accepted without performing any tests. While their motivation was also the removal of small stopping sets, their metric fails to pick up stopping sets such as those formed in Fig. 3 (suppose that $\eta = 1$), since it ignores the actual connections of a variable node. As an extreme case consider two variable nodes of degree $d > 2$ that share "all" their constraint nodes. Then all cycles that they participate in have an ACE value of $2(d-2)$. However these two nodes form a stopping set that would be picked up by the *Stopping-Set-Check* algorithm. These kinds of problems are more likely to occur at high degree nodes.

A joint approach where the ACE algorithm is used on low degree nodes and the *Stopping-Set-Check* on high degree nodes provides substantially improved results as explained in the next section.

IV. Simulations and Discussion

We used our algorithm to construct (603, 301) codes that have an irregular degree distribution obtained using density evolution based on the Gaussian Approximation [8]. The distributions are given by,

$$\begin{aligned} \lambda(x) &= 0.2186x + 0.1470x^2 + 0.1692x^4 \\ &\quad + 0.0136x^5 + 0.0517x^6 + 0.3999x^{19} \\ \rho(x) &= x^8 \end{aligned} \tag{6}$$

We constructed four different classes of codes -

- **Random** - The codes were chosen completely at random from the degree distribution ensemble.
- **Approx. Cycle EMD** - These codes were designed for the purposes of comparison with Tian et. al. Since the ACE technique reports significantly better performance than [2] we chose it as a benchmark. The codes have a $(d_{ACE} = 6, \eta = 3)$ profile. This means that all cycles in the code of size ≤ 12 have an ACE value at least 3

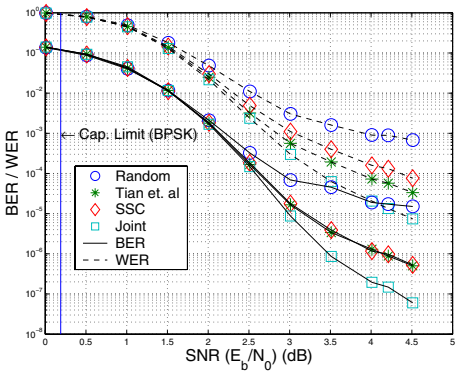

Fig. 5. AWGN Averaged Simulation results for (603,301) codes of each class

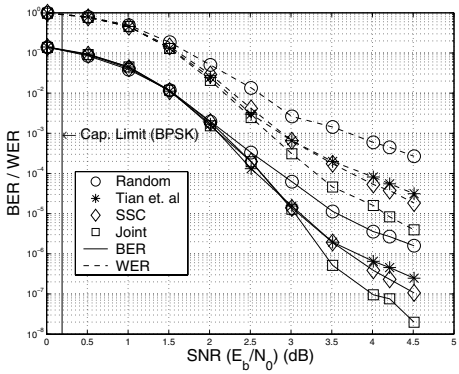

Fig. 6. AWGN Simulation Results of the best (603,301) codes of each class

(These parameters were chosen for illustrative purposes and higher values are also possible. [5])

- **Approx. *Stopping-Set-Check* (SSC)** - These were codes designed using our algorithm described in Section III. The parameters we chose for the code were only moderately optimized and there is room for more work here.
- **Joint** - In designing these codes we applied the ACE algorithm on the low degree nodes and our algorithm on the high degree nodes as discussed in section III-D. These codes provide a significant advantage at high SNR.

Since the construction process is inherently random, to provide a fair comparison, for each class we constructed 5 different codes. The performance was averaged over the best 4 codes for all the classes. The results are presented in Fig. 5. As far as Bit-Error-Rate (BER) goes our results (SSC) are almost identical to the results of Tian et. al, and are an order of magnitude better than randomly constructed codes. The joint approach performs substantially better, having a 0.5 dB gain at a BER of 10^{-6}. In terms of Word-Error-Rate (WER), the SSC method is slightly inferior to the method of Tian et. al, at a WER of 10^{-4} we lose by about 0.3-0.4 dB. However the Joint approach still remains about 0.5 dB better than Tian et. al. A comparison of the best codes found for each class is also made in Fig. 6. The Joint method is still clearly superior while there is hardly a difference between the SSC and Tian et. al approaches. At a BER of 10^{-6} the Joint code is about 3.2 dB from the Shannon limit which is quite good considering that the code is only of length 600.

Our approach to the code design problem has been to remove as many small stopping sets as possible. It is therefore natural to simulate these codes for the BEC to get an idea of the amount of stopping set reduction we obtain. These results are presented in Fig. 7. The WER of the "Joint" code construction is significantly lower than all the other constructions. This shows the absence of small stopping sets in the graph.

V. Conclusion

While asymptotic arguments have been made for the absence of small stopping sets in irregular LDPC code ensembles in the limit of large block length [7] a constructive technique that avoids them in actual constructions is needed from a practical perspective. This work presents a construction algorithm for the design of irregular LDPC codes at short block lengths that avoids small stopping sets. The performance of these codes is an order of magnitude better than randomly constructed codes. The method has a performance similar to the codes of Tian et. al [5]. Codes that are constructed using a combination of the two approaches seem to perform substantially better.

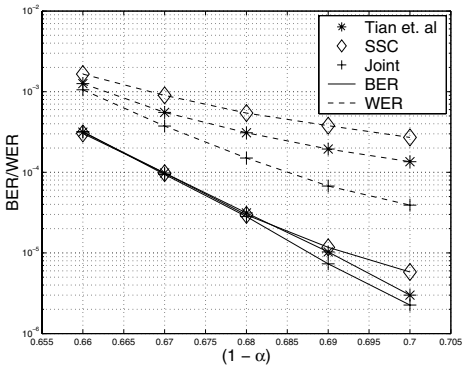

Fig. 7. BEC Averaged Simulation results for (603,301) codes of each class, $\alpha \rightarrow$ Channel Erasure Probability

References

[1] T. J. Richardson and R. L. Urbanke, "The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding," *IEEE Trans. on Info. Th.*, vol. 47, no. 2, pp. 599–618, 2001.

[2] Y. Mao and A. Banihashemi, "A Heuristic Search for Good Low-Density Parity-Check Codes at Short Block Lengths," in *IEEE Intl. Conf. Comm.*, 2001.

[3] X. Hu, E. Eleftheriou, and D.-M. Arnold, "Progressive Edge-Grown Tanner Graphs," in *IEEE GlobeCom*, vol. 2, 2001, pp. 995–2001.

[4] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. on Info. Th.*, vol. 48, no. 6, pp. 1570–1579, 2002.

[5] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular LDPC codes with low error floors," in *IEEE Intl. Conf. Comm.*, 2003, pp. 3125 –3129.

[6] ——, "Selective Avoidance of Cycles in Irregular LDPC Code Construction," *Accepted for IEEE Trans. on Comm.*

[7] D. Burshtein and G. Miller, "Asymptotic Enumeration Methods for analyzing LDPC Codes," *Submitted to IEEE Trans. on Info. Th.*

[8] S.-Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation," *IEEE Trans. on Info. Th.*, vol. 47, no. 2, pp. 657–670, 2001.